Opinnäytetyö (AMK)

Tietojenkäsittelyn koulutusohjelma

Yrityksen Tietojärjestelmät

2016

Alex Tamminen

LÄÄKETIETEELLISEN VERKKOSOVELLUKSEN HALLINTALIITTYMÄ

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittelyn koulutusohjelma

2016 | Sivumäärä: 41

Alex Tamminen

LÄÄKETIETEELLISEN VERKKOSOVELLUKSEN HALLINTALIITTYMÄ

Tässä opinnäytetyössä käsitellään lääketieteeseen liittyvän verkkosovelluksen hallintaliittymän kehitystä, eli ylläpitäjän näkymää sovelluksesta. Järjestelmään toteutetaan mahdollisuus lisätä, poistaa ja muokata sovelluksessa olevaa tietoa. Sovelluksessa tallennetaan tietoa liittyen sairauksiin ja oireisiin sekä näiden välillä olevia liitoksia. Tarkoituksena on kehittää mahdollisimman helppokäyttöinen, käytettävyydeltään hyvä sovellus.

Opinnäytetyö on toteutettu toimeksiantona Etsimo Oy:lle, joka luo hakukoneisiin ja hakukoneiden optimointiin liittyviä sovelluksia. Tutkimusmenetelmänä on käytetty kehittävää menetelmää.

Teoriaosuudessa käsitellään järjestelmän toteutuksessa käytettäviä ohjelmointikieliä. Lisäksi esitellään järjestelmään liittyvät ominaisuuksien vaatimukset. Käytännön osiossa esitellään järjestelmän toteutus ja toiminta.

Tutkimuksen tuloksena on käyttöönotettavissa sovellus lääketieteellisen sovelluksen ylläpitoon.

ASIASANAT:

Tietojärjestelmät, PHP, Angular 2 ohjelmakehys, MySQL, Sovelluksen hallintatyökalu

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Degree programme in Information Technology

2016 | Pages: 41

Alex Tamminen

MEDICAL WEB APPLICATION MANAGEMENT TOOL

This Thesis covers the creation of a medical web application management tool. It is meant for the administrator, where the administrator can easily manage adding, updating and deletion of symptoms and diseases, as well as the connections between them. The aim is to develop a user interface connected to a database, putting weight on the user interface, it being easy to use, keeping in mind the usability of the application.

This Thesis is created as per assignment for the company Etsimo ltd, which creates applications related to search engines and search engine optimization. The method used in this theses is development.

The theory part of this thesis covers the programming languages used in the development of the application as well as the requirements of the application.

The practical part of this thesis contains of presenting the creation of the application and presenting the usage of the application.

The result of this thesis is a working application for the management of a medical web application.

KEYWORDS:

Information Systems, PHP, Angular 2 software framework, MySQL, Application Management Tool

SISÄLTÖ

KUVAT

KUVIOT

KÄYTETYT LYHENTEET TAI SANASTO

CORS	Cross-Origin Resource Sharing. Mekanismi joka sallii resurssipyynnöt eri domeenien välillä.
CSS	Cascading Style Sheets, verkkosivuille kehitetty merkintäkieli. Määrittää dokumentin tyylin.
CSV	Comma-separated values. Tietokannan tallentaminen tekstimuodossa.
DOM	Document Object Model. Selaimeen ladattava html-dokumentti.
Elvis-operaattori	Notaatio: ”?”. Joissakin ohjelmointikielessä käytettävä notaatio, joka yhdistää kaksi lähdekoodissa olevaa elementtiä.
ES5	vastaa JavaScriptiä.
HTML	Hypertext Markup Language. Verkkosivujen merkintäkieli.
HTTP	Hypertext transfer protocol: protokolla, jota selaimet ja palvelimet käyttävät tiedonsiirtoon.
JSON	JavaScript Object Notation. Tallennusmuoto, jota käytetään lähinnä palvelimen ja sovelluksen väliseen tiedonkulkuun
MariaDB	Relaatiotietokantajärjestelmä
NPM	Node package manager. Node.js:n pakettienhallintajärjestelmä
PHP	PHP: Hypertext Preprocessor. Ohjelmointikieli.
RegExp	Regular Expression. Ohjelmoinnissa käytettävä säännöllinen lauseke.

SQL	Structured Query Language. Relaatiotietokantojen käsittelyyn käytettävä kieli.

1 JOHDANTO

Opinnäytetyön kehittämistyön tarkoitus ja tavoite on luoda toimiva, mahdollisimman helppokäyttöinen ja käytettävyydeltään myös mahdollisimman hyvä hallintaliittymä (hallintatyökalu) lääketieteellisen sovelluksen ylläpitoon.

Tämä opinnäyte on toteutettu toimeksiantona yritykselle Etsimo Oy. Yrityksen nimi viittaa etsimiseen, ja yritys luo hakukoneisiin liittyviä älykkäitä sovelluksia.

Tässä opinnäytetyössä kehitetty hallintaliittymä on lääketieteellistä sovellusta varten. Käyttäjä voi hakea sairauksia oireiden perusteella, tai oireita sairauksien perusteella. Sovelluksessa on käytetty lääketieteellistä tilastoa, jota on kerätty vuosien varrella. Tämän tapaisia sovelluksia löytyy myös muualta, mutta sovelluksesta tehdään mahdollisimman älykäs, kehittämällä algoritmeja ja logiikkaa.

Tähän sovellukseen tarvitaan myös hallintatyökalu, tai toisin sanoen hallintaliittymä, jossa sovelluksen ylläpitäjä voi muokata sovelluksessa olevaa dataa. Tämä opinnäytetyö liittyy kyseisen hallintaliittymän kehitykseen.

Sovelluksen kehityksessä on käytetty osittain ominaisuuksia ja algoritmeja, joita toimeksiantaja ei halua tulevan yleiseen tietoon, siitä syystä, että nämä erottavat kehitetyn sovelluksen muista samantapaisista sovelluksista. Tämä data on opinnäytetyössä joko jätetty mainitsematta, tai mainittu, että kyseessä on tietoa, jota en saa esittää.

Tämä opinnäytetyö kattaa ohjelmointikielten, Angular 2-ohjelmakehyksen ja kehitysympäristöjen (kehitysalustojen) esittelyn, siltä osin kuin nämä kehitettyyn sovellukseen liittyvät. Painopiste on Angular 2-ohjelmakehyksen esittämisessä, koska se on kehittämässäni sovelluksessani eniten käytetty. Näiden lisäksi esitellään sovelluksen suunnittelu ja valmiin sovelluksen esittely kuvilla ja lähdekoodi-kappaleilla.

Tämän opinnäytetyön teoriaosuuden lähteenä olen osittain käyttänyt Stackoverflow-sivustoa, jossa ohjelmoijat vastaavat toisten esittämiin kysymyksiin. Tämä lähde ei silloin välttämättä ole faktaa. Olen kuitenkin kriittisesti tulkinnut näitä vastauksia, verrannut vastauksia omaan tietämykseen, sekä ottanut huomioon miten monta henkilöä on merkinnyt annetut vastaukset hyviksi vastauksiksi. Tämä koskee tietysti myös muita sivustoja, jotka eivät ole ohjelmointikielten virallisia lähteitä.

2 JÄRJESTELMÄN KEHITYSTYÖKALUT

Tässä luvussa esitellään sovelluksen ohjelmointiin käytetyt ohjelmointikielet. Toimeksiantajalla ei ollut ennalta toivottuja ohjelmointikieliä, eli sain vapaasti valita ohjelmointikielet.

Järjestelmän kehittämiseen käytettiin selainpuolella Angular 2- ja TypeScript yhdistelmää, palvelinpuolen skriptaukseen PHP:ta. Tietokanta rakennettiin MySQL:llä. Palvelimena käytin Apachea, en lähde sitä kuitenkaan tarkemmin esittelemään, koska sovelluksen kehityksessä käytin Uniform Serveriä (lisää luvussa 4.2.), johon kuuluu Apache-palvelin.

Sovelluksen kehityksessä olisi voinut olla mahdollista yhdistää selainpuolen Angular 2 kommunikoimaan suoraan tietokannan kanssa, mutta tämä ei tietoturvallisuutta ajatellen olisi ollut lainkaan järkevää. Käyttäjä pystyisi erittäin todennäköisesti tekemään SQL-injektioita jos näin olisi tehty. Tämä johtuu siitä, että selainpuolen lähdekoodi on paremmin käyttäjän ulottuvuuksissa. (Stackoverflow 2016.) Siksi käytin myös PHP:tä sovelluksen kehittämiseen.

2.1 Angular 2

Angular 2 on varsin uusi ohjelmoakehys, ja aloittaessani järjestelmän kehittämisen, lopullinen versio oli juuri julkistettu. Angular 2 on AngularJS:n seuraaja. Virallinen Angular 2 versio julkistettiin syyskuun 15. päivä 2016. (Angular Blogspot 2016.) Angular 2 on suunniteltu käytettäväksi ensisijaisesti TypeScriptin kanssa, mutta vaihtoehtoisesti kehittämiseen voidaan käyttää JavaScriptiä tai Dartia (Angular 2016). Angular 2:sen kehittäjät kuitenkin vahvasti suosittelevat TypeScriptin käyttöä (Lerner ym. 2016, 60). Itse käytin TypeScriptiä sovelluksen kehittämiseen, näin ollen luvussa 2.1.2. esitellyssä esimerkkisovelluksessa on TypeScriptiä käytetty. Selaimissa toimivien sovelluksien lisäksi Angular 2 on myös suunniteltu mobiilisovellusten kehittämiseen (Tutorialspoint 2016.).

Angular 2 on rakennettu TypeScriptillä, mutta koska suurin osa selaimista eivät tällä hetkellä tue TypeScriptin ajoa, käännetään TypeScript ES5koodiksi, eli normaaliksi JavaScriptiksi. Tämä tapahtuu automaattisesti ajon aikana. (Lerner ym. 2016, 59–60.)

Angular 2-projektin tekemiseen suositellaan erittäin vahvasti node.js-alustaa. (Lisää node.js:tä luvussa 3.1.) Jos halutaan kuitenkin olla käyttämättä node.js:ää, se on mahdollista, mutta vaatii paljon konfigurointia. Se ei ole ideaalinen tilanne muun muassa suorituskykyä ajatellen ja lisäksi Angular 2 käyttää valtavan määrän paketteja, jolloin näiden importointi hankaloituisi huomattavasti. (Stackoverflow 2016.) Näin ollen valitsin sovelluksen rakentamiseen node.js-alustan.

Aloitin Angular 2 ohjelmoinnin Eclipsen kehitysympäristössä, mutta ainakaan tässä vaiheessa siinä ei ole Angular 2 tukea, eli se teki ohjelmoinnista erittäin hankalaa. Siksi siirryin käyttämään Visual Studio Code:a, joka on Eclipsen tapaan vapaan lähdekoodin ohjelmisto. Tämä osoittautui hyväksi valinnaksi, ja tätä suositeltiin myös monessa paikassa Angular 2-projektin käyttöön.

2.1.1 Angular 2-sovelluksen arkkitehtuuri

Angular 2:n arkkitehtuuri voidaan Tutorialpointsin ja Angular 2 virallisten sivujen mukaisesti jakaa seuraaviin osiin:

- Moduulit
- Komponentit
- Mallit (engl. Template)
- Metadata
- Datan sidonta (engl. Data Binding)
- Palvelut (engl. Service)
- Direktiivit (engl. Directive)
- Riippuvuuksien injektointi (engl. Dependency Injection)

Angular 2-sovellus rakentuu moduuleista, joissa on komponentteja. Komponentit ovat luokkia, ja nämä merkitään ”export”-notaatiolla, jotta komponentteja voidaan kutsua myös toisissa komponenteissa. (Tutorialspoint 2016.) Juurikomponenttina on sovellus itsessään, eli HTML-dokumentti. Komponentit ovat koottavia, eli suuria komponentteja voidaan koota pienemmistä komponenteista. Angular 2-sovellus on siis komponentti, jonka selain hahmottaa sovelluksen latautuessa. Koska kyseessä on komponenttipuu, juurikomponentti lataa kaikki puussa olevat muut komponentit. (Lerner ym. 2016, 74.) Komponentti merkitään @Component-notaatiolla. Yksi komponentti vastaa yhtä DOM-elementtiä. Ilman @Component-notaatiota kyseessä on komponentin sijaan vain

tavallinen luokka. Tämä notaatio on yksi esimerkki metadatasta ja myös samalla direktiivistä. Metadataa ja direktiivejä on huomattava määrä, ja tämä tunnistetaan aina alkavan @-merkillä. (Tutorialspoint 2016 & Angular 2016.) Näistä tärkein on juuri mainitsema @Component-notaatio, koska ilman sitä meillä ei ole lainkaan toimivaa sovellusta.

Template (suom. malli) on sovelluksen selaimessa näkyvä osa. Angular 2:sen malli on täysin dynaaminen. Malliin merkitään html-sisältö, tai määritellään polku erilliseen html-tiedostoon. Komponentissa käytetään joko `template: ` HTML-koodisi tähän `` tai `templateUrl: 'html-tiedosto.html'`. (Tutorialspoint 2016 & Angular 2016.) Jälkimmäisessä esimerkissä html-koodi on sijoitettu erilliseen tiedostoon, ensimmäisessä se on lisätty komponentti-tiedostoon. Huomioitavaa on, että toisessa käytetään hipsukoita ja toisessa heittomerkkejä.

Datan sidonta tarkoittaa, että datan lähde sidotaan html-näkymään. Angularissa on neljä päätyyppiä datan sitomiseen. (Tutorialspoint 2016.) Yhteistä näillä kuitenkin on se, että komponentit ja html-näkymä sidotaan jollain tapaa, joko DOM:sta komponenttiin, komponentti DOM:n tai molempiin suuntiin. (Angular 2016.) Kaksisuuntainen sidonta on Angular 2:sen erikoisominaisuuksia, eli siitä syystä nostan tämän tyypin esille. Kaksisuuntaisessa sidonnassa käytetään syntaksia `[(ngModel)]` (Angular 2016.). Tämä on tarkemmin esiteltynä luvussa 2.1.3, jossa olen nostanut esille joitakin kehittämässäni sovelluksessani käyttämäni syntaksia.

Angularissa palvelu-komponentit eivät juurikaan eroa muista ohjelmointikielistä, joissa ilmenee palveluita, eli siitä syystä en esittele niitä sen tarkemmin. Palveluilla on yleensä jokin tietty tehtävä, kuten esimerkiksi http-kutsujen tekeminen (Angular 2016). Näin olen myös kehittämässäni sovelluksessa palveluita käyttänyt.

2.1.2 Angular 2-sovelluksen konfigurointi ja esimerkkisovellus

Angularin virallisilla www-sivuilla (www.angular.io) mainitaan seuraavalla sivulla olevat komponentit ja konfigurointitiedostot, joita tarvitaan siihen, että voidaan aloittaa Angular 2-sovelluksen kehittämisen.

Sen jälkeen kuin node.js ja npm ovat asennettuina, luodaan seuraavat konfigurointitiedostot:

- package.json: ilmoitetaan tarvittavat npm-riippuvuudet
- tsconfig.json: määritellään, miten kääntäjä tuottaa JavaScriptiä projektin TypeScript -tiedostoista
- typings.json: lisäkirjastoiden määrittely, joita TypeScript ei automaattisesti tunnista
- systemjs.config.js: määritellään sovelluksen moduulien sijainnit käytettäväksi moduulien lataajalle

Seuraavaksi esittelen yksinkertaisesen Angular 2 -sovelluksesta ja siihen tarvittavat tiedostot kuvineen. Lisää tietoa Angular 2:sen arkkitehtuurista voi lukea edellisessä luvussa, 2.1.1.

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: `<h1>Hello {{name}}</h1>`
})
export class AppComponent { name = 'Angular'; }
```

Kuva 1. app.component.ts-tiedosto. Lähde: (www.angular.io)

Ensin luodaan usemmiten pääkomponenttina toimiva, app.component.ts, johon sovelluksen sisältö rakentuu. Tiedoston nimeksi voi tietysti valita itselleen mieluisan nimen.

Selector viittaa html-elementtiin, johon komponentin sisältö ladataan. Malli (engl. template) on varsinainen html lähdekoodi, eli sovelluksen niin sanottu sisältö. (Angular 2016.) Kuvassa 1 olevassa app.component.ts-tiedostossa selaimeen tulostetaan yksinkertaisesti ”Hello Angular”. Aaltosulut ovat osa Angular 2:sen syntaksia. Angular 2:n syntaksista voi lukea seuraavassa luvussa, 2.1.3.

Yleensä app.component.ts-tiedostossa ei näytetä ”template”-kohdassa varsinaista html-koodia, vaan tähän laitetaan `<router-outlet></router-outlet>` notaatio, joka tarkoittaa että, tähän reititetään sovelluksen näkymät, eli komponentit (Tutorialspoint 2016). Tässä

esimerkissä on kuitenkin kyseessä sovellus, jossa on vain yksi komponentti, AppComponent, eli tässä tilanteessa tämä on hyvä ratkaisu.

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent }  from './app.component';

@NgModule({
  imports:      [ BrowserModule ],
  declarations: [ AppComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

Kuva 2. app.module.ts-tiedosto. Lähde: (www.angular.io)

”app.module.ts” on sovelluksen päämoduuli, jossa ilmoitetaan kaikki sovelluksessa luodut komponentit ja moduulit. Kuvassa 2 olevien ”imports”, ”declarations” ja ”bootstrap”:n lisäksi mainitaan yleensä ”providers”, johon merkitään sovelluksessa käytetyt palvelumoduulit. (Angular 2016.) Lisäksi monisivuisissa sovelluksissa on yleensä luotu myös reitityksiä, nämä merkitään reititysmoduuliin, joka sitten merkitään ”imports” kohtaan.

Kuvan 2 pienessä esimerkkisovelluksessa tarvittava moduuli on vain selainmoduuli (engl. browser module), koska kyseessä on selainpohjainen sovellus. (Angular 2016.) Kuten mainitsin luvun alussa, Angular 2 on myös suunniteltu mobiilisovelluksien kehittämiseen, eli selainpohjaisissa sovelluksissa pitää erikseen importoida selainmoduuli. Tässä esimerkkisovelluksessa app.module.ts-tiedostossa ilmoitetaan myös kuvassa 1 luotu komponentti (AppComponent) ja lopuksi esiladataan sovellus (engl. bootstrap).

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule }              from './app.module';

platformBrowserDynamic().bootstrapModule(AppModule);
```

Kuva 3. main.ts-tiedosto. Lähde: (www.angular.io)

Sovellus käynnistetään main.ts-tiedostosta. Tähän importoidaan sovelluksen päämoduuli, eli AppModule, joka käynnistää kaikki sovelluksen muut osat (Angular 2016.).

Lopuksi luodaan index.html tiedosto. Tälle sivulle pitää lisätä tärkeimpinä osina konfigurointitiedosto systemjs.config.js-tiedoston sijainti sekä main.ts -tiedoston sijainti, jotta sovellus toimii. Jotta selaimeen tulostuu sovelluksen sisältö, lisätään index.html-tiedostoon body-kappaleen sisälle app.component.ts-tiedostossa määritelty html-elementin nimi, eli tässä esimerkisovelluksessa ”my-app”. (Angular 2016.)

```
<body>
    <my-app> Your app here! </my-app>
</body>
```

Projektin konfigurointia ei ole pakko itse tehdä. Angular 2:sen virallisilta sivuilta saa kloonata GitHub-projektin, jossa kaikki konfiguroinnit ovat jo valmiiksi tehty, mikä tarkoittaa, että voi heti aloittaa sovelluksen rakentamisen.

2.1.3 Angular 2 syntaksi

Tässä on esiteltynä joitakin esimerkkejä, jotka kuuluvat Angular 2 ominaisuuksiin. Olen rajannut nämä niihin, joita olen kehittämässäni sovelluksessa käyttänyt. Kaikkia en ole ottanut huomioon, nämä ovat vain muutamia esimerkkejä. Luvussa 5.3, jossa esittelen yhden lähdekoodiesimerkin, esitellään lisää Angular 2:sen ominaisuuksia. Kaikista syntakseista ja ominaisuuksista löytyy tietoa Angular 2:sen virallisilta sivuilta (www.angular.io).

```
<p>{{user.name}}</p>
```

Aaltosuluilla tulostetaan selaimeen komponentissa olevan muuttujan arvo. Tässä tapauksessa kyseessä on user-objektin nimen (engl. name) arvo.

```
<input [(ngModel)]='name' [ngModelOptions]='{standalone: true}'>
```

ngModel sitoo automaattisesti käyttäjän syöttämän arvon komponentissa olevaan samannimisen muuttujan arvoon. Tässä kohtaa ngModel:n arvoa ei ole komponentissa ennalta määritelty, eli tässä joudutaan lisäämään vielä ngModelOptions, joka sallii, että name saa olla irrallisena ja ilman ennalta määriteltyä arvoa.

```
<button (click)=”doSomething()”>Click Me!</button>
```

Angular 2:ssa on lukuisa määrä DOM-tapahtumien käsittelijöitä, yllä yksi esimerkki, jossa kutsutaan metodia ”doSomething()” kun käyttäjä napsauttaa painiketta. Nämä tapahtumien syntaksi on hyvin lähellä esimerkiksi JavaScriptin syntaksia. Erona on, että JavaScriptissä syntaksi samalle käskylle olisi `(onClick)`.

```
ngOnInit() { }
```

ngOnit metodiin kirjataan yleensä ne metodit, joita halutaan suoritettavan heti kun sivulle navigoidaan. On mahdollista jättää käyttämättä ngOnInit metodia, ja merkitä suoritettavat metodit luokan konstruktoriin, mutta tämä ei ole suositeltavaa. ngOnInit-metodin lisäksi pitää luokassa merkitä, että komponentin pitää implementoida kyseisen metodin. Tämä onnistuu syntaksilla: `export class MyComponent implements OnInit`.

```
<p *ngIf=”show”> Hello World! </p>
```

Viimeisenä esimerkkinä esittelen *ngIf:n. Yllä voidaan olettaa että “show” on Boolean, jolloin p-elementin sisältö tulostuu selaimeen vain, jos “show” on tosi, eli sama kuin ”true”.

2.2 PHP

PHP on suosittu verkkosovelluksien toteutukseen tarkoitettu ohjelmointikieli. W3Techsin mukaan noin 82% verkossa olevista sovelluksista käyttää PHP:tä palvelinpuolella, mikä antaa viitettä juuri siihen, miten suosittu ohjelmointikieli PHP on. PHP on alun perin kehitetty helpottamaan dynaamisten toimintojen lisäämistä verkkosivuille aikana, jolloin työkaluja kyseiseen tarkoitukseen ei juurikaan ollut. Kielen ensimmäiset versiot julkaistiin jo vuonna 1995. (Mainiotech 2016.) PHP on avoimen lähdekoodin skriptauskieli. Monet suuret yritykset käyttävät PHP:tä, muun muassa Facebook ja WordPress. (w3schools 2016.)

PHP:tä käytetään useimmiten palvelinpuolen skriptaukseen. Näin myös käytin PHP:tä opinnäytetyöhöni liittyvän sovelluksen kehittämisessä. Palvelinpuolen skriptauksen lisäksi PHP:tä voi käyttää myös komentorivillä, mutta silloin se vaatii parserin. On myös mahdollista käyttää PHP:tä työpöytä-sovelluksien kehittämiseen, tämä on kuitenkin harvinaista, koska PHP ei ensisijaisesti ole tarkoitettu käyttöliittymän tekoon. (PHP

2016.) En lähde tarkemmin kertomaan kahdesta jälkimmäisistä tavoista käyttää PHP:tä, koska se ei liity tähän opinnäytetyöhön.

PHP:ssä on integroituna tuki moniin suosittuihin tietokantoihin, esimerkkeinä mainittakoon MySQL, PostgreSQL ja Oracle (PHP 2016.). Tämä on suuri etu, kun skriptissä voi, ilman lisäasetuksia tai asennuksia, käyttää komentoja tietokantaan liittyen. Kehittämässäni sovelluksessa pystyin php-skripitissä esimerkiksi tekemään SQL-kyselyn MySQL-tietokantaan helposti komennolla `mysqli_query`. Lisää MySQL:stä seuraavassa luvussa 2.3.

PHP:tä suoritetaan palvelimella ennen verkkosovelluksen lähettämistä selaimelle, mikä tarkoittaa, että PHP ei vaadi tukea selaimelta, ja sillä voi käsitellä esimerkiksi palvelimen tiedostoja ja tietokantoja. PHP:n käyttäminen vaatii palvelimen, jossa on mahdollista suorittaa PHP-koodia. (Ohjelmointiputka 2016.) Palvelin on sovellus, joka toimittaa www-sivuja. Palvelimella, joka käyttää PHP:tä, käy php-skripti niin sanotun PHP-prosessorin läpi, jossa serveri hakee tiedostosta PHP:n avaustageja `<?php`. Kun tälläinen löytyy, se prosessoidaan PHP-moduulissa. Prosessointi loppuu kun PHP:n lopputagi `?>` löytyy. (Suehring & Valade 2013, 272)

PHP ei ole sidottu mihinkään tiettyyn palvelimeen. PHP ei myöskään ole sidottu tiettyyn käyttöjärjestelmään, vaan toimii tunnetuimmilla käyttöjärjestelmillä; Windows, Linux, Mac OS ja suurimmassa osassa Unix-järjestelmiä (PHP 2016.).

PHP-skriptit voidaan luoda erillisinä tiedostoina tai HTML-lähdekoodin tiedostoon. Seuraavalla sivulla on yksinkertainen esimerkki PHP-skriptistä HTML-tiedostossa, joka tulostaa selaimeen ”Hello World”. PHP skriptit kirjoitetaan `<?php ?>` -elementin sisälle.

```
<!DOCTYPE html>
<html>
<body>
<h1> My first PHP page <h1>
<?php
echo ‘Hello World’
?>
</body>
</html>
```

PHP-skripti voi PHP-koodin lisäksi sisältää HTML:ää, CSS:ää ja JavaScriptiä (w3schools 2016.).

Editorina PHP:tä varten käytin Eclipsen kehitysympäristöä ja ajoittain myös Notepad++:aa.

2.3 MySQL

PHP:n käytön yhteydessä MySQL on suosituin relaatiotietokantajärjestelmä, jota verkkosovelluksissa käytetään. Se ajetaan palvelimella ja sopii sekä pienille että suurille sovelluksille. Siinä käytetään SQL-standardeja ja sen saa ladata ilmaiseksi käyttöönsä. MySQL:ää käyttävät monet suuret yritykset, muun muassa Facebook, Twitter ja Wikipedia. (w3schools 2016.)

MySQL-järjestelmä muodostuu tietokannoista, jotka sisältävät tauluja. Järjestelmää hallinnoidaan MySQL-palvelimella, sekä muilla taustalla ajettavilla apusovelluksilla (Suehring & Valade 2013, 449). Kun tässä luvussa puhun palvelimesta niin viittaan MySQL-palvelimeen.

Kun halutaan luoda uusi tietokanta, lähetetään komento palvelimelle, joka luo uuden polun tietokannalle tarvittavine tiedostoineen. Jotta tämä onnistuisi, pitää palvelimen ensin olla käynnissä. Palvelin konfiguroidaan yleensä sellaiseksi, että se on käynnissä niin kauan kun tietokone, jossa palvelin pyörii, on käynnissä. Näin ollen palvelin on yhtäjaksoisesti käynnissä ja vain odottaa käskyjä. Yhdellä palvelimella voidaan hallinnoida useita tietokantoja. (Suehring & Valade 2013, 449-450)

Tietokannassa oleva data tallennetaan yhteen tai useampaan tauluun. Ensin luodaan tyhjä tietokanta ja tyhjät taulut, ennen kuin dataa voidaan tietokantaan syöttää. (Suehring & Valade 2013, 450) Taulut koostuvat kentistä ja riveistä. Tauluissa voi olla yksi, monta tai ei yhtään tietuetta. Jos taulussa ei ole yhtään tietuetta, taulu on tyhjä. (Ohjelmointiputka 2016.)

Taulujen nimeämisessä pitää muistaa se, että nimi kannattaa olla kuvaava. Nimi saa sisältää kirjaimia, numeroita, alaviivoja ja dollarimerkkejä, mutta ei välilyöntejä (Suehring & Valade 2013, 477).

Alla on esimerkki taulusta, jonka nimeksi voisi antaa ”users”. Esimerkkitaulussa on yksi tietue, eli yksi rivi dataa, joka tässä tapauksessa koostuu kentistä ”id”, ”firstname” ja ”lastname”, ja joilla on vastaavasti arvot ”1”, ”Alex” ja ”Tamminen”. Suehring & Valade

suosittelevat, että taulun nimi olisi yksikkömuodossa, eli tässä tapauksessa ”user”, mutta tämä on makuasia.

```
+----+-----------+----------+
| id | firstname | lastname |
+----+-----------+---------+
|  1 | Alex      | Tamminen |
+----+-----------+----------+
```

Kenttiä luotaessa niille määritellään yleensä nimi, tietotyyppi, maksimipituus, onko tieto pakollinen ja mahdollinen oletusarvo. Kentän nimeksi kannattaa valita selkeä ja kuvaava nimi, sekä välttää välilyöntejä, erikoismerkkejä ja skandinaavisia kirjaimia. Maksimipituutta määriteltäessä pitää olla erityisen tarkka; lähes aina kannattaa varata mieluummin liian paljon tilaa kuin liian vähän. (Jyväskylän yliopisto 2016.)

Mahdollisimman moni kenttä kannattaa myös määritellä pakolliseksi tai käyttää oletusarvoa. Tietotyyppi voi olla numeerinen, alfanumeerinen, päivämäärä ja/tai kellonaika, looginen tai bittijono. (Jyväskylän yliopisto 2016.) Kaikkein yleisin näistä tietotyypeistä, on merkkijono, toisin sanoen alfanumeerinen tietotyyppi (Suehring & Valade 2013, 481). Kehittämässäni sovelluksessa käytin vain alfanumeerisia ja numeerisia tietotyyppejä, näin ollen en esittele muita tietotyyppejä kuin näitä kahta.

Alfanumeeriisen tietotyypin pituudeksi voidaan valita kiinteä maksimipituus, jolloin kyseessä on ”CHAR”-tyyppi. Voidaan sanoa että kentän tyyppi voisi olla CHAR(10), joka tarkoittaa, että kentällä on 10 merkin kiinteä pituus. Jos syötetty arvo on yli 10 merkkiä, vain ensimmäiset 10 merkkiä huomioidaan. Jos arvo on vaikka vain 5 merkkiä, niin seuraavat 5 merkkiä ovat tyhjiä, mutta kentän kokonaispituus on siitä huolimatta 10. Tätä kannattaa miettiä tietokantaa suunniteltaessa. Jos etukäteen tietää, että kentässä olevat arvot tulee olemaan samanpituisia (tai vain vähän pituudeltaan eriäviä), kannattaa käyttää kiinteää maksimipituutta. Muissa tapauksissa kannattaa vain määritellä maksimipituus, eli tyypiksi ”VARCHAR”, ja jos merkkijono on lyhyempi kuin maksimipituus, merkkijono ottaa vain sen verran tilaa kuin se tarvitsee. (Suehring & Valade 2013, 481-483)

Numeerinen tietotyyppi rakentuu numeroista. Tietokantaan on mahdollista tallentaa kokonaislukuja (INT) sekä desimaalilukuja (DECIMAL). Näillä voidaan suorittaa esimerkiksi laskutoimituksia. Jos alusta asti on kuitenkin selvää, että dataa ei käytetä laskutoimituksiin, voidaan nämä numerot yhtä hyvin tallentaa merkkijonoina, koska

ohjelmoija käyttää näitä ohjelmakoodissaan siinä tapauksessa todennäköisesti vain merkkijonoina. (Suehring & Valade 2013, 482-483)

Jokaiselle taululle pitää määritellä perusavain (engl. primary key), joka yksilöi taulun sisältämät tietueet. Kahta samanlaista riviä ei saisi olla. (Suehring & Valade 2013, 479) Jokaisella tietueella pitää olla yksilöllinen, eli uniikki perusavain. Perusavain muodostetaan yhdestä tai useammasta taulun kentästä, jälkimmäisessä tapauksessa perusavainta kutsutaan yhdistetyksi avaimeksi (engl. composite key). Perusavain ei saa puuttua tai olla NULL. NULL tarkoittaa tuntematonta arvoa, eli sen arvo voisi olla mikä tahansa, tai ei mitään. (Jyväskylän yliopisto 2016.)

Tauluissa voi myös esiintyä viiteavaimia. Viiteavaimella voidaan viitata jonkin toisen taulun avaimeen ja näin ollen luodaan relaatioita taulujen välillä. (Suehring & Valade 2013, 480) Relaatioista voi lukea enemmän luvussa 3.2, jossa esittelen oliomallin luomastani tietokannasta.

Kaikki kommunikointi tietokannan kanssa tapahtuu välittämällä viestejä palvelimelle. Tämä kommunikointi tapahtuu standardoiduilla SQL-kyselyillä. Kyselyt muistuttavat paljon selkokielistä englantia. Kyselyn ensimmäinen sana on verbi, joka kuvailee mitä käyttäjä haluaa tehdä. Esimerkkejä ovat ”INSERT” (suom. lisää) ja ”DELETE” (suom. poista), jotka siis tarkoittavat datan lisäämistä tai poistamista tietokannasta. Toinen yleinen komento on ”SELECT” (suom. valitse). Kyselyyn kuuluu myös ”FROM”-sana ja usein ”WHERE”. FROM:lla kerrotaan mistä taulusta halutaan dataa hakea, WHERE:n kohdalla voidaan antaa ehtoja. WHERE ei kuitenkaan ole pakollinen. (Suehring & Valade 2013, 450-451) Näiden lisäksi on vielä muitakin komentoja.

Esimerkki yksinkertaisesta select-kyselystä:

```
SELECT id, firstname, lastname FROM users
```

Kysely palauttaa kaikki käyttäjien id:t, etunimet ja sukunimet users-taulusta, tietue tietueelta. Saman kyselyn voi myös kirjoittaa lyhyemmin *-merkillä, sen sijaan, että kaikki kentät kirjoitetaan erikseen. Palautuva data on kuitenkin sama kuin yllä olevassa kyselyssä. Seuraavalla sivulla olevassa esimerkissä on kuitenkin lisätty vielä ehto `WHERE id = 1`, toisin sanoen, haetaan kaikki käyttäjät, joilla id on yhtä kuin 1.

```
SELECT * FROM users WHERE id = 1
```

Tässä voisi olettaa, että jokaisella käyttäjällä on uniikki id, eli se tarkoittaisi sitä, että kysely palauttaa joko ei mitään, tai yhden tietueen. Tämä kysely voisi siis palauttaa esimerkiksi: `1, Alex, Tamminen`.

Ennen kuin sain kehittämässäni sovelluksessa käyttööni tietokannan, jossa todellista lääketieteellistä dataa, loin tietokannan phpMyAdminillä, joka on osa Uniform Serveriä (lisää luvussa 4.2.) Tämä mahdollisti sen, että tietokannan pystyi luomaan ja muokkaamaan käyttöliittymällä komentorivin sijaan.

3 JÄRJESTELMÄN SUUNNITTELU

Projektin aluksi järjestettiin palaveri, jossa kartoitettiin lähtötilanne ja järjestelmän vaatimukset. Kyse on lääketieteeseen liittyvästä sovelluksesta, jossa voidaan hakea mahdollisia sairauksia oireiden perusteella tai mahdollisia oireita sairauden perusteella. Työnjakoon tehtiin selvä rajaus: Toinen henkilö luo niin sanotulle ”tavalliselle” käyttäjälle näkyvän osuuden sovelluksesta, minä ylläpitäjälle näkyvän sovelluksen.

3.1 Järjestelmän vaatimukset

Ensimmäinen vaatimus oli salasanasuojattu sisäänkirjautuminen varsinaiseen hallintaliittymään. Alkuperäisiin vaatimuksiin kuului luoda käyttäjätaulu, jossa on vain id, käyttäjänimi ja salasana. Ylimääräistä aikaa kuitenkin jäi, eli päätin lisätä (tulevaisuutta ajatellen) mahdollisuuden käyttäjälle päivittää tietonsa (etunimi, sukunimi, sähköposti) ja sen lisäksi käyttäjä pystyy vaihtamaan salasanansa.

Vaatimuksiin kuului, että sovelluksessa tulee voida lisätä, poistaa ja muokata yksittäisiä sairauksia ja oireita sekä näiden välisiä liitoksia. Lisää näistä liitoksista kuvassa 4 seuraavassa luvussa ja luvussa 5.3. Jokaisella sairaudella ja oireella tulee olla oma näkymä, jossa sairautta tai oiretta voidaan muokata tai poistaa kokonaan. Lisäksi näkymässä luetellaan oireeseen liitettyjä sairauksia tai sairauteen liitettyjä oireita. Näitä voidaan poistaa tai muokata. Käyttäjä pystyy myös lisäämään liitoksia sairauksien ja oireiden välillä, sekä oireiden ja sairauksien välillä.

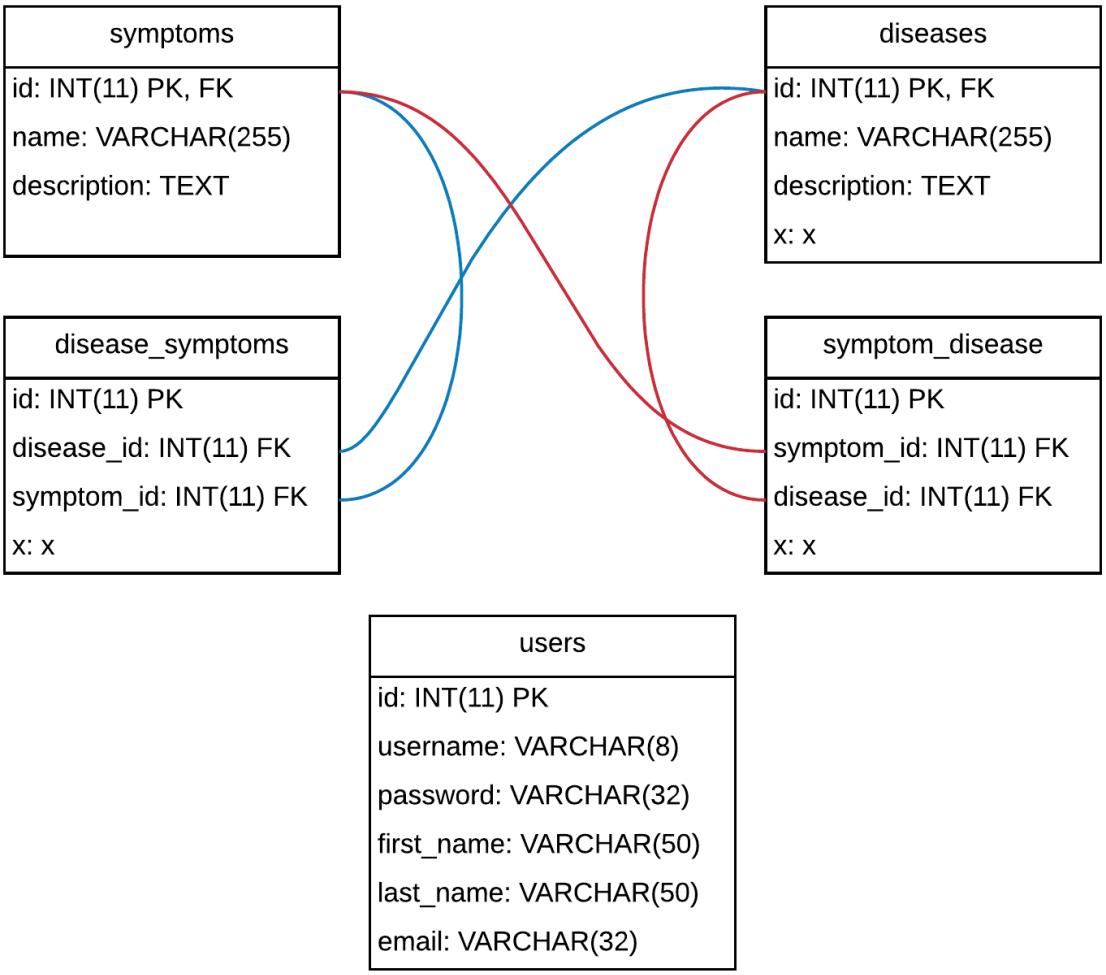
Sairauksien löytäminen useammalla oireella kuului myös vaatimuksiin. Tämä tarkoittaa, että käyttäjä voi syöttää monta oiretta, ja jos näillä oireilla on yhteisiä sairauksia, nämä näytetään käyttäjälle.

Tässä opinnäytetyössä ei ole otettu huomioon tietoturvallisuusaspektia.

3.2 Tietokannan suunnittelu

Tietokannan suunnitteluun sain erittäin tarkat ohjeet, koska kehittämäni sovellus on liitettynä toiseen sovellukseen, eli niin sanotulle tavalliselle käyttäjälle näkyvään osuuteen. Toisin sanoen, tämä siis tarkoittaa että tietokanta on yhteinen. Aloitin kuitenkin

ensin rakentamalla oman tietokannan pienellä määrällä testausdataa. Myöhemmin projektin aikana sain käyttööni isomman määrän todellista testausdataa. Kuva 4 esittää tietokannan oliomallin taulut liitoksineen. Tauluissa on käytetty lyhenteitä ”PK” ja ”FK”. ”PK” on lyhenne sanasta ”primary key”, eli perusavain. ”FK” on lyhenne sanasta ”foreign key”, eli viiteavain.



Kuva 4. Oliomalli.

Tietokanta koostuu viidestä taulusta. Käyttäjät ovat tallennettu ”users”-tauluun. Käyttäjät viittaavat tässä sovelluksessa sovelluksen ylläpitäjiin. Taululla ei ole liitosta muihin tauluihin. Taulun perusavain on uniikki (käyttäjä) id.

Oliomallissa voidaan nähdä, että ”diseases”-, ”disease_symptoms”- ja ”symptom_disease”-tauluissa on kenttä ”x”. Tämä on tietoa, jota toimeksiantaja ei halua tulevan yleiseen tietoon.

Sairaudet löytyvät ”diseases”-taulusta, jossa perusavain on uniikki id, joka on samalla myös viiteavain, jota käytetään liitostauluissa. ”symptoms”-taulu on melkein identtinen ”diseases”-taulun kanssa, mutta siinä ei ole ”x”-kenttää.

Käytössä on myös kaksi liitostaulua, ”symptom_disease” (suom. oire_sairaus) ja ”disease_symptoms” (suom. sairaus_oireet). ”symptom_disease”-tauluun on tallennettu oireisiin liittyviä sairauksia. Vastaavasti ”disease_symptoms”-tauluun on tallennettu sairauksiin liittyviä oireita.

4 KEHITYSYMPÄRISTÖ

Lähdekoodi-editoreina käytin luvussa 2.1 ja 2.2 mainittuja Visual Studio Codea ja Eclipseä sekä ajoittain Notepad++:aa. Kuten mainitsin luvussa 2.1, on hyvin suositeltavaa käyttää node.js alustaa Angular 2:sen kanssa. Myös PHP vaatii kehitysalustan, ja kehityksessä käytin minulle ehdotettua Uniform Serveriä, jossa on yhdistettynä palvelin, MySQL sekä PhpMyAdmin.

4.1 Node.js-kehitysympäristö

Node.js on avoimen lähdekoodin alustariippumaton palvelinpuolen kehitysympäristö, joka on kirjoitettu JavaScriptilla. Ryan Dahl kehitti Node.js:n vuonna 2009. Node.js on avoimen lähdekoodin kehitysalusta, joka toimii Windowsilla, Mac OS:lla ja Linuxilla. Node.js tarjoaa monia JavaScript-moduuleita, jotka perustuvat JavaScript-kirjastoihin. Muun muassa eBay, Yahoo, Microsoft ja PayPal käyttävät Node.js:ää. (Tutorialspoint 2016.)

Node.js:ssä luodut lähde-tiedostot sisältävät lähdekoodia. Nämä ovat yleensä JavaScript tiedostoja, eli ovat ”.js”-päätteisiä (Tutorialspoint 2016). Kehitetyssä sovelluksissani lähdetiedostot ovat ”.ts”-paatteisiä, eli TypeScript-tiedostoja. Nämä käännetään kuitenkin puhtaaksi JavaScriptiksi ajon aikana.

Node.js on täysin asynkroninen, eli monia pyyntöjä voidaan esimerkiksi suorittaa samanaikaisesti (Node.js 2016.). Tämä tarkoittaa, että sovelluksen suoritus on nopeampaa kuin synkronoidun. Synkronoidussa suorituksessa suoritetaan ohjelmakoodi siinä järjestyksessä kuin se on kirjoitettu, yksi rivi kerrallaan ja seuraava rivi odottaa että aiempi rivi on suoritettu. Asynkronissa ohjelmakoodissa seuraava rivi voidaan ohjelmakoodin rakennelmasta riippuen aloittaa, ennen kuin aiempi rivi on suoritettu valmiiksi, sekä eri järjestyksessä kuin ohjelmakoodi on kirjoitettu. (Stackoverflow 2009.)

Node.js:n mukana tulee myös npm, eli Node.js Package Manager. Sillä voidaan ladata verkosta Node.js-paketteja projektiin. Npm toimii suoraan komentoriviltä ja npm-paketin asennus suoritetaan komennolla: `npm install paketin_nimi`. Paketti ladataan node_modules hakemistoon. Kun paketti on ladattu ja asennettu, sen voi ottaa käyttöön

omassa sovelluksessa. (Pilvikoodarin Blogi 2016.) Omassa projektissani käytin muutamia kolmansien osapuolten npm paketteja, lisää näistä luvussa 5.7.

4.2 Uniform Server

Uniform Server on kevyt ja pienikokoinen vapaan lähdekoodin WAMP-palvelin Windows käyttöjärjestelmää varten. Uniform Server on myös siirrettävä, eli sitä voidaan käyttää esimerkiksi USB-tikulta. (Uniform Server 2016.)

WAMP on lyhenne sanoista ”Windows, Apache, MySQL ja PHP”. ”P” voi PHP:n sijaan myös tarkoittaa Perliä tai Pythonia, mutta kun käytöstä puhutaan, PHP on näistä yleisin. Apache-palvelin on oleellisin osa WAMP:a. (TechTerms 2013.) Palvelin on sovellus, joka ajetaan tietokoneella. Kaikista palvelimista, Apache on suosituin (Suehring & Valade 2013, 13). Apachea käytetään Uniform Server:ssä palvelimen ajoon lokaalisti paikallisesti Windows alustalla, joka tarkoittaa, että sovellusta voidaan testata selaimessa (TechTerms 2013.).

Apachea kehutaan vakaaksi ja ongelmat Apache-palvelimen kanssa ovat harvinaisia. Koska Apache on vapaan lähdekoodin sovellus, voidaan tätä muokata sopivaksi omien vaatimuksien mukaisesti. (Suehring & Valade 2013, 14)

Uniform Serverissä on integroituna myös phpMyAdmin, joka on selaimen kautta käytettävä MySQL-tietokannan hallintatyökalu. Käyttöliittymästä voidaan muun muassa luoda tietokanta tauluineen ja tehdä sql-kyselyitä sen sijaan, että nämä komennot suoritettaisiin suoraan komentoriviltä. (phpMyAdmin 2016.)

phpMyAdmin on avoimella lähdekoodilla kehitetty työkalu, jossa käytetty PHP:ta työkalun kehittämiseen. phpMyAdmin on tarkoitettu MySQL- ja MariaDB-tietokantojen hallintaan. phpMyAdminilla pystyy importoimaan kokonaisia tietokantoja CSV- ja SQL-muodossa. Sama pätee myös tietokantojen viennissä. Tämän lisäksi voidaan esimerkiksi luoda graafinen PDF-tiedosto tietokannan rakenteesta. (phpMyAdmin 2016.)

4.3 Testiympäristön pystyttäminen

Ensialkuun asensin node.js:n Windows-koneelleeni. Asennus oli helppo ja mutkaton. Node.js:n virallisilta sivuilta (www.nodejs.org) saa ladattua tarvittavan asennustiedoston.

Node.js:n mukana sai asennettua myös kehitettävään sovellukseen tarvittavan npm:n. Uniform Server oli myös helppo asentaa virallisilta sivuilta (www.uniformserver.com) saadun asennustiedoston kanssa.

Angular 2- ja PHP-yhdistelmä ei ainakaan vielä ole täysin tavallinen ratkaisu, koska Angular 2:sta suositellaan node.js-alustalle. Tässä tapauksessa tämä tarkoitti sitä, että minulla oli kaksi erillistä palvelinta (node.js ja Uniform Server) ja pyyntöjen tekeminen näiden välillä ei ole oletuksena sallittua, tietoturvariskien takia. (Spring 2016.) Tämä ongelma ilmeni vain kehitysvaiheessa, joten domain välisen kommunikoinnin ratkaisin asentamalla Chrome-selaimeen laajennuksen ”CORS”, joka sallii domeenien väliset pyynnöt.

Uniform Serverin kanssa ilmeni myös ensialkuun ongelmia Apache-palvelimen käynnistymisessä. Ongelmaksi osoittautui koneelleni asennettu Skype, joka käyttää oletuksena samaa porttia kuin Uniform Server. Ongelman ratkaisin muuttamalla Skypen oletusporttia.

Loin tietokannat phpMyAdminillä, joka on integroituna Uniform Serveriin. Lisäsin tauluihin pienen määrän testausdataa. Tietokannan tyypiksi valitsin MyISAM:n sijaan InnoDB:n, koska InnoDB tukee viiteavaimia (MySQL 2016.). Käyttämässäni tietokannassa tarvitsin viiteavaimia luodakseni viitteet oireiden ja sairauksien liitostauluun, sekä sairauksien ja oireiden liitostauluun. Tietokannan rakenteen voi nähdä kuvassa 4 luvussa 3.2.

Kun ensimmäinen toimiva sovellus oli pystyssä näillä tauluilla, sain käyttööni tietokannan tauluineen, jossa on oikeaa dataa, jolloin sovelluksen testaukseen oli myös tarpeeksi dataa.

5 SOVELLUKSEN KEHITYS JA TOIMINTA

Tässä luvussa esittelen sovelluksen kehitystä. Kehityksessä on käytetty Angular 2-TypeScript yhdistelmää, PHP:tä ja MySQL:ää. Angular 2:n ja PHP:n kommunikointi on 4Devin suositteluiden mukaisesti ratkaistu siten, että selainpuolen ohjelmakoodissa tehdään http-pyyntöjä erillisiin php-skripteihin.

Kehittämässäni sovelluksessa PHP:n käyttö rajoittui siihen, että PHP-skripteissä tehtiin vain kyselyitä tietokantaan, palauttaen tietokannasta pyydettyä dataa selainpuolen lähdekoodiin, eli Angular 2- ja Typescriptin lähdekoodiin. Toisin sanoen voidaan sanoa, että PHP toimi vain välikätenä tietokannan ja Angular 2- ja TypeScript-koodin välillä, muuhun tarkoitukseen PHP:tä ei tarvittu.

Luvussa 5.1. oleva kuva 5 esittää yhden luomani php-skriptin. Kaikki sovelluksessa olevat php-skriptit ovat hyvin samankaltaisia, jossa PHP-skripti vastaanottaa tiettyä dataa, jota tarvitaan SQL-kyselyissä. Tämän jälkeen SQL-kysely suoritetaan ja frontendiin palautetaan haluttua dataa. Tämä palautettava data on joko ”success” tai ”NULL” siinä tapauksessa, että halutaan tietää, esimerkiksi sisäänkirjautumisen yhteydessä, onko käyttäjätunnus ja salasana olemassa. Jos pyynnöt koskevat datan noutoa tietokannasta, palautetaan data suurimmaksi osaksi taulukkona JSON-muodossa. Jos noudetaan vain yksi tietue tietokannasta, tämä tietue palautetaan JSON-objektina.

5.1 Sisäänkirjautuminen

Koska käyttäjät kuuluivat järjestelmän vaatimuksiin, ensimmäinen toteutettu näkymä oli sisäänkirjautumislomake. Käyttäjän syötettyä käyttäjätunnuksen, salasanan ja painettua login-painiketta tarkistetaan Angular 2-metodissa ensin, että lomakkeen kentät on täytetty ennen kuin käyttäjätunnus ja salasana lähetetään PHP-skriptiin. Esittettelen tässä http-kutsun lähdekoodin, kuten myös login.php-skriptin.

```
// request to server to authenticate user
login(id, username, password) {
    let body = new URLSearchParams();
    body.set('username', username);
    body.set('password', password);

    let headers = new Headers();
    headers.append('Content-Type', 'application/x-www-form-urlencoded');
    let options = new RequestOptions({ headers: headers });

    return this.http.post(this.userUrl, body.toString(), options)
        .map((res: Response) => {
            let resp = res.json();
            if (resp != null) {  // user is authenticated
                localStorage.setItem('currentUser', JSON.stringify({
                    id: resp.id, username: username, firstName: resp.firstName,
                    lastName: resp.lastName, email: resp.email
                }));
                return true;
            }
            else {
                return false; // user not authenticated
            }
        });
}
```

Kuva 4. Sisäänkirjautumisen http-kutsu

PHP-skriptistä palautuu frontendiin joko sana ”success” tai NULL, joiden perusteella määritellään, onko käyttäjätunnus ja salasana valideja. Jos käyttäjä on autentikoitu, eli palautettu data on eri kuin NULL, laitetaan käyttäjä väliaikaiseen muistiin myöhempää käyttöä varten. Lisäksi palautetaan kutsuvalle metodille ”true”, jos käyttäjä on autentikoitu. Näin ollen käyttäjä reititetään kutsuvassa metodissa varsinaiseen hallintaliittymän alkunäkymään, muutoin käyttäjälle näytetään virheilmoitus, että joko käyttäjätunnus tai salasana on virheellinen. Tietokannassa salasana ja käyttäjätunnus on tehty merkkikokoriippuvaiseksi, eli kenttä on taulussa merkitty latin7_general_cs:ksi.

Kuvassa 5 login.php-skriptissä tiedot ajetaan mysqli_real_escape_string –funktion läpi välttääksemme SQL-injektioita. Seuraavaksi tarkistetaan yhteys tietokantaan ennen kuin SQL-kysely suoritetaan. Lopuksi suoritetaan SQL-kysely, jossa tarkistetaan, onko käyttäjää olemassa. Jos SQL-kysely palauttaa yhden tietueen, tämä tarkoittaa, että käyttäjä on autentikoitu ja silloin palautetaan frontendiin sana ”success”.

```
if($_SERVER["REQUEST_METHOD"] === "POST") {
    //prevents SQL injection
    $username = mysqli_real_escape_string($conn,$_POST['username']);
    $password = mysqli_real_escape_string($conn, $_POST['password']);

    // Check connection
    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error);
    }

    //query to be executed
    $query = "SELECT *
              FROM users
              WHERE username = '$username' and password = '$password'";

    $result=mysqli_query($conn, $query);

    //if the result is just one row, the user has credentials
    if (mysqli_num_rows($result) == 1) {
        echo json_encode("success");
    }

    else {
        echo json_encode(null);
    }
}
```

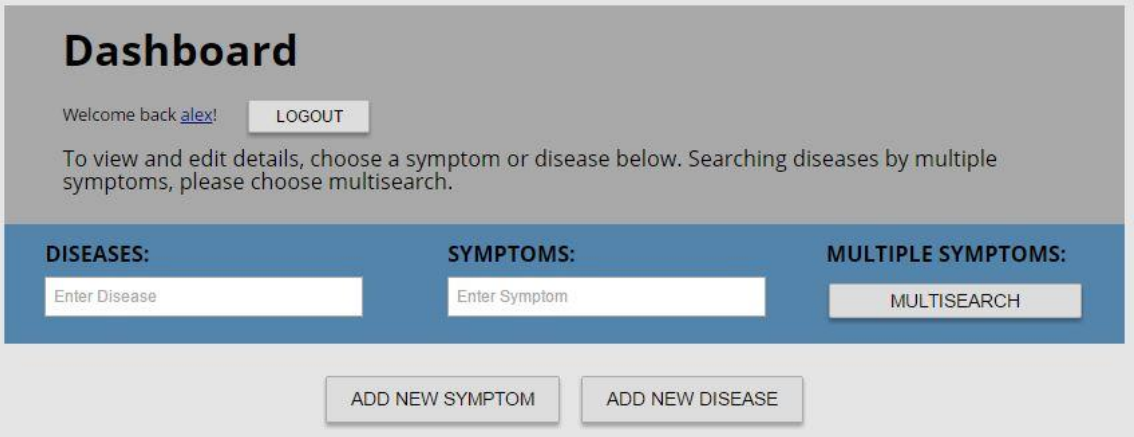
Kuva 5. login.php

5.2 Sairaudet ja oireet

Seuraavaksi toteutettiin näkymä sairauksien ja oireiden hallintaa varten. Seuraavalla sivulla oleva kuva 6 esittää näkymän sisäänkirjautumisen jälkeen. Kyseessä on myös samalla pääkomponentti, suurin osa sovelluksen muista osista ovat lapsi-komponentteja ja samalla myös toteutettu lapsireitityksinä. Valitsin ratkaista komponentit ja reitit tällä tavalla, koska reitittäminen uudelle sivulle esimerkiksi käyttäjän valitessa oireen, ei olisit ollut järkevää, varsinkaan käytettävyyttä ajatellen.

Koska tämä on isäkomponentti, se tarkoittaa, että päänäkymä on aina esillä sivun yläosassa riippumatta siitä missä käyttäjä navigoi. Poikkeuksena on sisäänkirjautumisnäkymä ja myös niin sanottu käyttäjätilin näkymä, jossa käyttäjä voi katsoa ja muokata omia tietojaan. Lisää jälkimmäisestä näkymästä luvussa 5.6.

Tässä isäkomponentissa on pudotusvalikot sairauksille ja oireille ennakoivalla tekstinsyötöllä, sekä painikkeet uuden sairauden ja oireen lisäämiseen. Näkymässä on myös painike ”Multisearch”, jolla voidaan hakea sairauksia useamman oireen perusteella. Lisää tästä ominaisuudesta luvussa 5.6.



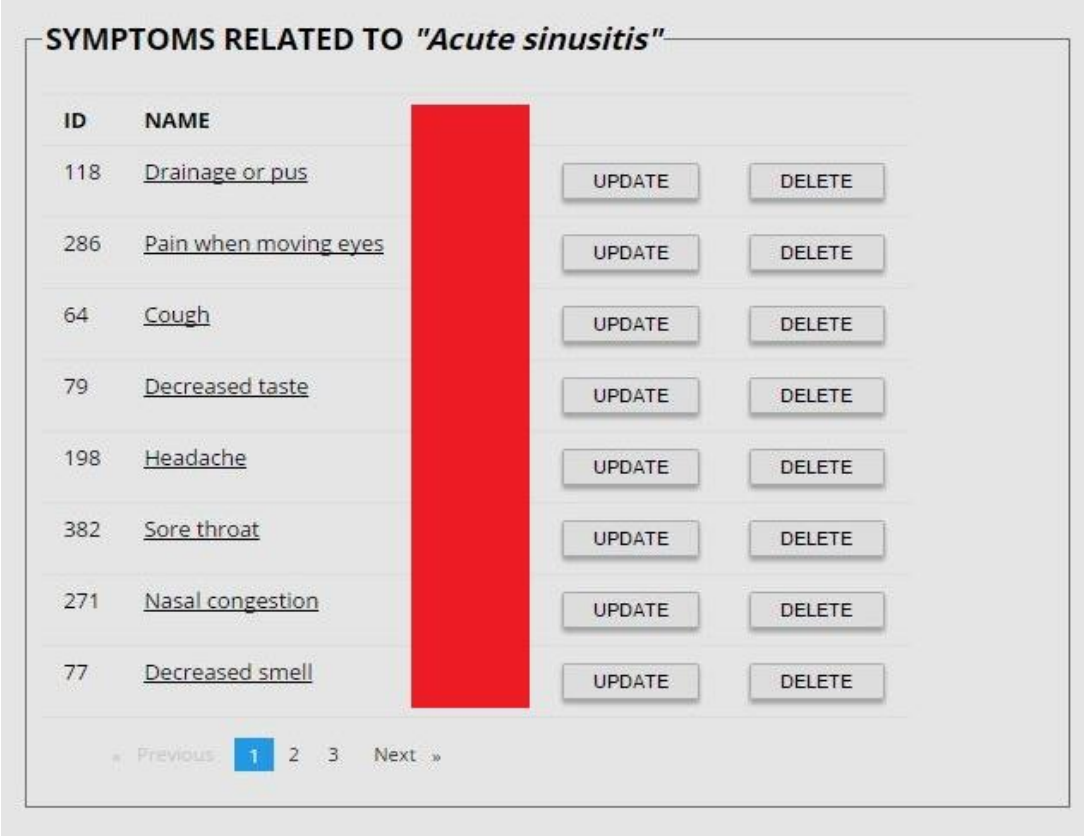
Kuva 6. Pääkomponentti

Kun käyttäjä valitsee sairauden tai oireen pudotusvalikosta, avautuu näkymä, jossa voidaan muokata sairauksien ja oireiden nimeä ja kuvausta. Sairauksilla on näiden lisäksi yksi numeerinen arvo, joka on myös muokattavissa. Muokkauksen lisäksi käyttäjä voi poistaa koko sairauden tai oireen tietokannasta. Tämä on kuitenkin todellisuudessa erittäin epätodennäköistä, joten tähän olen luonut varmistuksen, jotta säästyttäisiin virhepainalluksilta. Jos käyttäjä haluaa poistaa oireen tai sairauden, häntä pyydetään antamaan käyttäjätunnuksensa ja salasanansa, jotka tarkistetaan tietokantaa vastaan ennen kuin sairaus tai oire poistetaan.

5.3 Oireiden ja sairauksien sekä sairauksien ja oireiden väliset liitokset

Valitun oireen näkymään tulostetaan myös siihen liittyviä sairauksia. Sama koskee sairauksia, valitun sairauden näkymään tulostetaan selaimeen siihen liitettyjä oireita. Seuraavalla sivulla olevassa kuvassa 7 on valittu sairaus ”Acute sinusitis” (suom. poskiontelotulehdus), johon luetellaan tietokannassa olevia oireita, jotka ovat liitettynä poskiontelotulehdukseen. Kuvassa punaisella merkityllä alueella on numeerinen arvo, jota en saa esitellä.

Käytettävyyden parantamiseksi käyttäjä voi järjestää luettelon joko liitoksen id:n, oireiden nimien tai numeerisen arvon mukaan, laskevassa tai nousevassa järjestyksessä. Lisäksi lueteltuja oireita voidaan myös napsauttaa ja siten siirtyä kyseisen oireen näkymään.



Kuva 8. Oireiden ja sairauksien liitokset

Näitä olemassa olevia liitoksia voidaan myös päivittää tai kokonaan poistaa. Jos käyttäjä haluaa poistaa liitoksen, ”delete”-painikketta napsautettuaan käyttäjälle näytetään ponnahdusikkuna, jossa varmistetaan, että käyttäjä todellakin haluaa poistaa olemassa olevan liitoksen. Poiston jälkeen luettelo liitoksista päivitetään.

Käyttäjä pystyy myös luomaan uusia liitoksia sairauksien ja oireiden välillä sekä oireiden ja sairauksien välillä. Samassa näkymässä, jossa olemassa olevat liitokset ovat sijoitettuna, on myös pudotusvalikko ennakoivalla tekstinsyötöllä, jossa on lueteltuna kaikki oireet, joilla ei vielä ole liitoksia kyseiseen sairauteen, tässä tapauksessa, poskiontelotulehdukseen. Valittuaan oireen, avautuu näkymä, jossa käyttäjä pystyy lisäämään aiemmin mainitun numeerisen arvon. Tallennettuaan uuden liitoksen, liitoksien luettelo päivittyy.

Seuraavalla sivulla on esimerkki SQL-kyselystä tietokantaan, jossa noudetaan ”symptoms_disease”-taulusta ne oireet, joilla ei ole liitosta sairauteen, tässä tapauksessa sairaus, jolla id on sama kuin 4:

```
SELECT s.*
FROM symptoms s
WHERE NOT EXISTS
            (SELECT 4
            FROM symptoms_disease sd
            WHERE sd.symptom_id = s.id AND sd.disease_id = 4);
```

Esitelläkseni Angular 2:n ominaisuuksia, alla oleva lähdekoodi vastaa oireiden pudotusvalikkoa. Tässä koodinkappaleessa ei ole huomioitu ennakoivaa tekstinsyöttöä.

```
<select [(ngModel)]="sel_symptom" (ngModelChange)="onSelectConnection(sel_symptom)">
     <option *ngFor="let symptom of symptoms" [ngValue]="symptom">
          {{symptom.name}}
     </option>
</select>
```

”ngModel” sitoo valitun oireen “sel_symptom” objektiin. ”ngFor”:n tarkoitus tässä on iteroida läpi taulukon. Tässä tapauksessa on noudettu tietokannasta kaikki oireet ja tallennettu ne ”symptoms”-taulukkoon. ”{{symptom.name}}”-komennolla käsketään, että pudotusvalikossa näytetään yksittäisen oire-objektin nimen arvo. ”ngValue” puolestaan sitoo koko ”symptom” objektin ”sel_symptom” objektiin, jotta koko objekti on käytössä, pelkän oireen nimen sijaan. Luvusta 2.1.3. voi lukea lisää Angular 2 syntaksista.

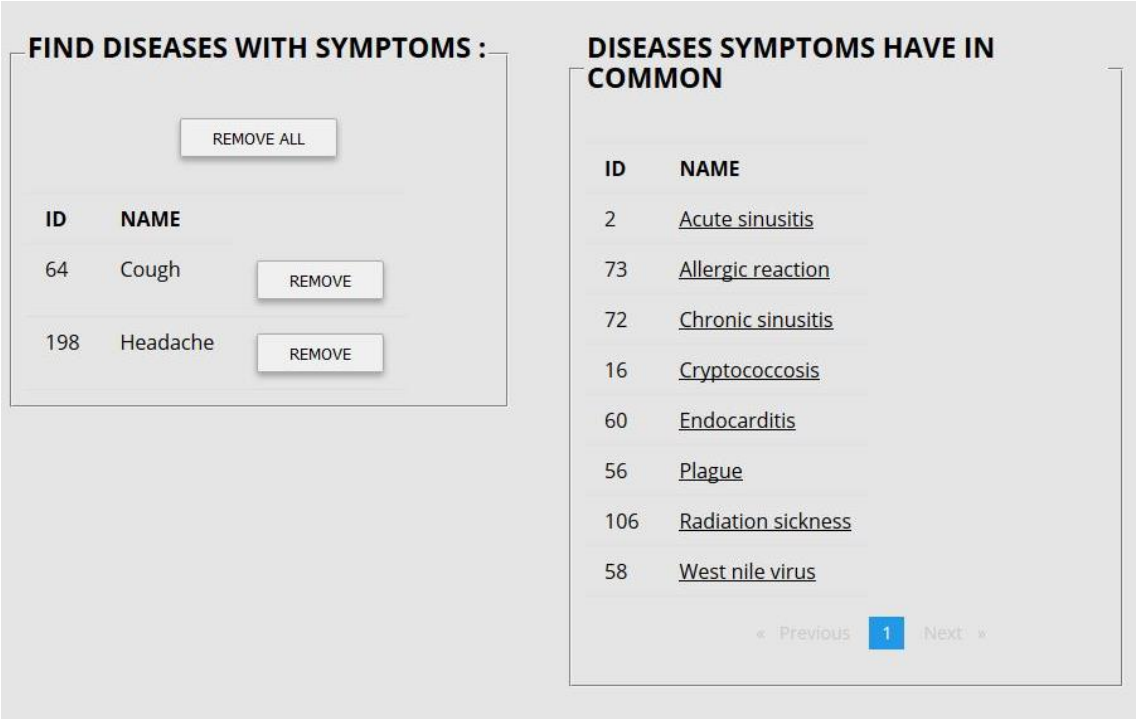
”ngModelChange” odottaa, että käyttäjä valitsee pudotusvalikosta oireen ja tämän tapahtuessa kutsuu metodia ”onSelectConnection” parametrinaan valittu ”sel_symptom”-objekti. Metodi avaa muutoin piilossa olevan näkymän, jossa luodaan uusi liitos valitun sairauden ja oireen välillä.

5.4 Sairauksien haku useamman oireen perusteella

Käyttäjä pystyy hakemaan sairauksia useamman oireen perusteella. Tämä tapahtuu pudotusvalikosta ennokoivalla tekstinsyötöllä, sen jälkeen kuin käyttäjä on hallintaliittymän pääkomponentista napsauttanut ”Multisearch”-painiketta. Päätin lisätä tämän painikkeen, sen sijaan, että pudotusvalikko olisi esillä pääsivulla. Syy tähän on, että jos käyttäjä navigoi sovelluksessa muilla sivulla, hän pystyy helposti palamaan aiempaan suoritettuun hakuun.

Kun käyttäjä valitsee pudotusvalikosta oireen, käyttäjälle näytetään ne sairaudet, joissa valitut oireet tai oire ilmenevät. Seuraavalla sivulla olevassa kuvassa 8 on oireet ”cough” (suom. yskä) ja ”headache” (suom. päänsärky) valittu. Oikealla puolella kuvaa on

lueteltuna ne sairaudet, joissa ilmenee sekä yskä että päänsärky. Nämä valitut oireet tallennetaan lokaalisti session ajaksi, jotta käyttäjä pystyy palaamaan viimeiseen suoritettuun hakuun, siitä huolimatta vaikka navigoisi sovelluksen muissa osioissa.



Kuva 9. Sairauksien haku monen oireen perusteella

Sairauksien lista päivittyy reaaliaikaisesti sen mukaan kuin käyttäjä tekee valintoja, eli valitsee lisää oireita, tai poistaa listalta oireita.

Alla on esimerkki SQL-kyselystä, jossa haetaan kahden oireen perusteella yhteisiä sairauksia ”symptom_disease”-taulusta. Oireilla on id:t 2 ja 3. Tulokset järjestetään sairauksien nimien (dis.name) perusteella nousevassa (ASC) järjestyksessä, toisin sanoen aakkosjärjestyksessä.

```
SELECT dis.*
FROM symptom_disease
JOIN diseases dis ON dis.id = symptom_disease.disease_id
WHERE symptom_id IN (2, 3)
GROUP BY dis.name
HAVING COUNT (DISTINCT symptom_id)= 2
ORDER BY dis.name ASC
```

Tässä hakunäkymässä on myös ajateltu käytettävyyttä. Ennakoivan tekstinsyötön kohdalla käyttäjä voi lisätä oireen listaan painamalla enter-näppäintä (hiirellä napsauttamisen lisäksi), ja valinnan jälkeen kursori pysyy valintakentässä. Tämän lisäksi

käyttäjä pystyy järjestämään sairaudet (tuloslistan) id:n tai nimen perusteella, nousevassa ja laskevassa järjestyksessä. Sairautta voi myös napsauttaa ja siirtyä kyseisen sairauden näkymään.

5.5 Oireen tai sairauden lisäys

Käyttäjä pystyy luomaan uuden sairauden tai oireen napsauttamalla joko ”Add New Symptom”- (suom. ”Lisää Oire”) tai ”Add New Disease” (suom. ”Lisää Sairaus”)-painiketta. Silloin käyttäjä reititetään näkymään, jossa on lomake uuden sairauden tai oireen tietojen täyttämiseen. Reitityksessä lähetetään parametrina joko ”symptoms” tai ”diseases”. Tätä tarvitaan siihen tarkoitukseen, että tiedetään näytetäänkö näkymässä numeerisen arvon kenttä, koska vain sairaudella on tämä kenttä (kts. Kuva 4 luvussa 3.2.). Jos parametri on ”diseases”, merkitään boolean ”showNumeric” todeksi, joka näyttää numeerisen arvon kentän.

Kuvassa 7 on esitettynä metodi oireen tai sairauden lisäämiseen. Lähdekoodia on muokattu hieman, muuttujat ”showNumeric” ja ”numeric” ovat itse keksittyjä.

```
saveNewMedical() { // saves new symptom / disease
    // checks if name has value and is not only white space(s). If so, error msg is displayed
    if (!/\S/.test(this.name) || !this.name) {
        this.showModal('front'); // helper method for error msg
    }
    else { // name value is valid
        // means that numeric is shown in template, therefore a new disease is to be added
        if (this.showNumeric === true) {
            if (isNaN(this.numeric)) { // checks if  valid number, if not, it's set as null
                // 0 is id, will be checked in backend and proper id will be set there
                let disease = new Disease(0, this.name, this.description, null);
                this.helperSave('diseases', disease); // saves new disease
            }
            else { // diseaseame has valid numeric and will be saved
                // 0 is id, will be checked in backend and proper id will be set there
                let disease = new Disease(0, this.name, this.description, this.numeric);
                this.helperSave('diseases', disease); // helper method that saves new disease
            }
        }
        else { // do not have numeric value, means it is a symptom to be added
            let symptom = new Symptom(0, this.name, this.description);
            this.helperSave('symptoms', symptom); // saves symptom
        }
    }
}
```

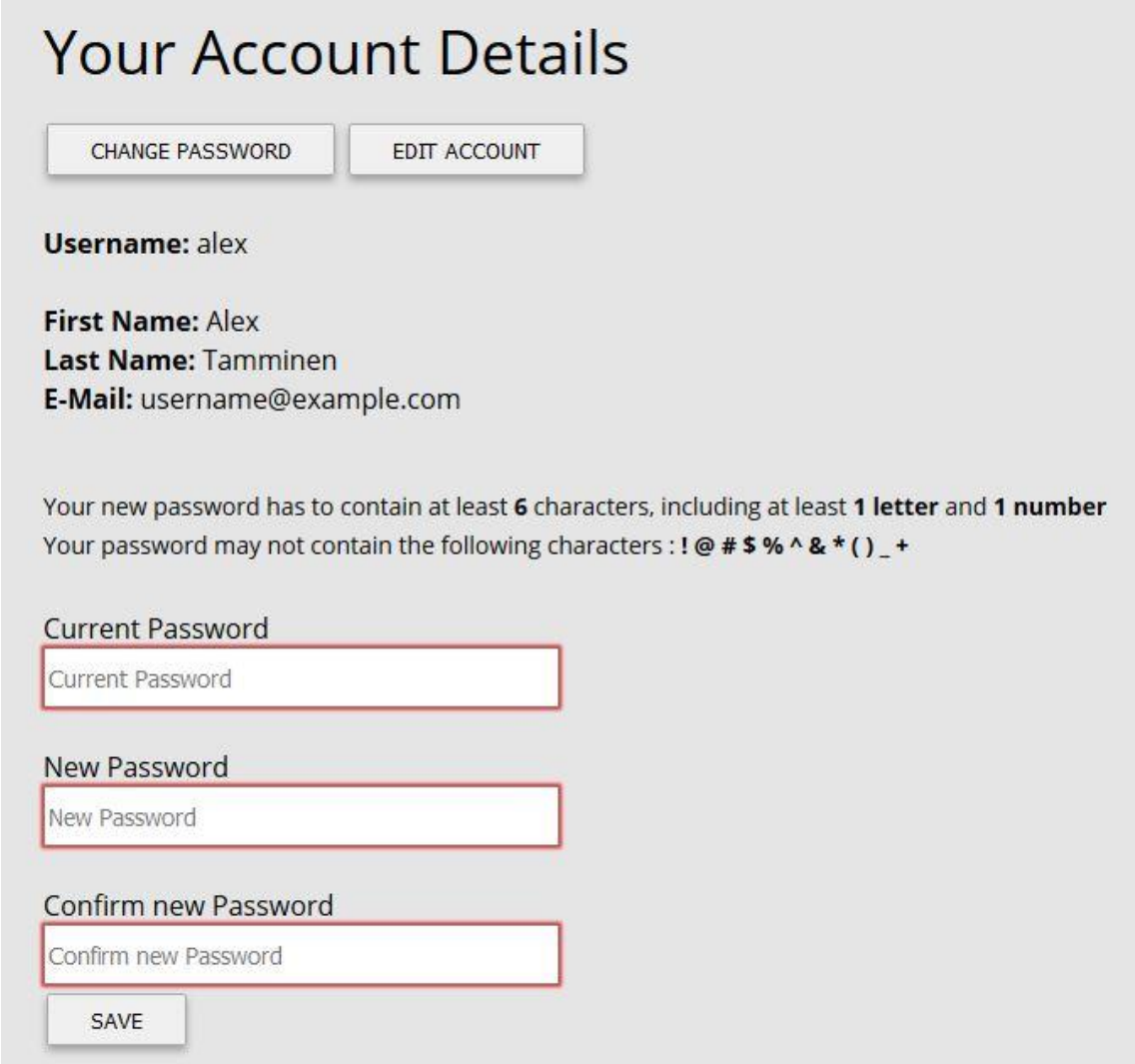
Kuva 7. Uuden sairauden tai oireen lisäys

Ensimmäiseksi tarkistetaan, ettei käyttäjä ole syöttänyt nimeksi pelkkiä välilyöntejä, jonka jälkeen tarkistetaan onko boolean ”showNumeric” totta, jos näin on, niin kyseessä on uuden sairauden lisäys. Tämän jälkeen tarkistetaan ”numeric”, jos käyttäjä on syöttänyt vahingossa vaikka kirjaimen, laitetaan oletusarvoksi null, muutoin tallennetaan käyttäjän syöttämä arvo. Sairaudelle annetaan tässä vaiheessa id 0, tämä tarkistetaan PHP-skriptissä, jossa annetaan sairaudelle oikea id. Tämän jälkeen kutsutaan apumetodia ”helperSave” tässä tapauksessa merkkijonolla ’diseases’, jotta PHP-skriptissä tiedetään, että kyseessä on sairaus, jolloin se tallennetaan ”diseases”-tauluun. Tämän lisäksi parametrina tulee tietysti käyttäjän syöttämät tiedot. ”helperSave”-metodi kutsuu ”medical-service”-komponentissa olevaa http-kutsua. Kaikki http-kutsut, käyttäjään liittyviä http-kutsuja lukuunottamatta, löytyvät ”medical-service”-komponentista.

5.6 Käyttäjän tiedot

Järjestelmän vaatimuksiin ei tämän opinnäytetyön puitteissa kuulunut käyttäjän tiedot ja näiden muokkaus. Ylimääräistä aikaa kuitenkin jäi, ja tulevaisuutta ajatellen näitä tietoja kuitenkin todennäköisesti tarvitaan, ainakin osittain. Päätin siis luoda näkymän, jossa käyttäjä pystyy muokkaamaan tietojaan ja vaihtamaan salasanansa. Kuvassa 9 on esitelty tämä näkymä sen jälkeen kun käyttäjä on napsauttanut ”vaihda salasana”, eli ”Change Password”-painiketta.

Salasanan vaihdon yhteyteen on lisätty validointia. Salasana pitää olla vähintään kuuden merkin pituinen, sisältäen vähintään yhden kirjaimen ja yhden numeron. Muutamat erikoismerkit eivät ole sallittuja. Tietokannassa on merkkikoko myös huomioitu. Sähköpostin muuttamisessa on myös lisätty validointi, jotta varmistetaan, että syötetty sähköposti on validi sähköpostiosoite.



Kuva 10. Käyttäjätiedot

5.7 Kolmannen osapuolen Angular 2-moduulit

Angular 2-projektin kehitykseen on luotu jonkun verran npm-paketteja, moduuleita, joita saa ilmaiseksi ottaa käyttöönsä. Yksittäiset yksityishenkilöt tai ryhmä -henkilöitä ovat yleensä nämä luoneet. Näillä paketeilla pystyy helposti lisäämään ominaisuuksia omaan projektiinsa. Ihan helpolla näissä ei kuitenkaan välttämättä pääse, koska moduulit on luotu oletustoiminnallisuuksilla ja -ulkoasuilla, jotka moduulin kehittäjä/kehittäjät ovat tehneet sellaiseksi kuin ovat halunneet. Tämä tarkoittaa, että moduuleita täytyy mahdollisesti muokata sellaisiksi, että vastaavat oman projektin tarpeita. Näin kävi tässä sovelluksessa, sekä ulkoasuja ja toiminnallisuuksia piti muokata jonkun verran.

Tässä sovelluksessa käytin kolmea kolmansien osapuolten moduulia. Tärkein näistä oli ennakoiva tekstinsyöttö. Toinen oli ponnahdusikkunat, joita näytetään käyttäjälle virheen tapahtuessa. Olisi tietysti ollut mahdollista käyttää sovelluksessa alert-ponnahdusikkunaa, eli `window.alert(”virheilmoitus tähän”)`, mutta tämän ulkoasun muokkaus ei ole mahdollista, mikä ei tietenkään ole ideaalinen tilanne.

Kolmas moduuli, jota käytin oli sivutus. Tätä tarvittiin esimerkiksi siinä, kun luetellaan oireeseen liittyviä sairauksia välttääksemme liian pitkää listaa. Tämä moduuli jäi ainoaksi, jota ei tarvinut paljon muokata. Ominaisuuksiltaan tämä oli toimiva sellaisenaan, vain ulkoasua joutui hieman muuttamaan.

6 TULOKSET

Tämän opinnäytteen tavoitteena oli toteuttaa lääketieteellisen sovelluksen hallintatyökalu, jossa sovelluksen ylläpitäjä pystyy muokkaamaan tietokantaan tallennettua tietoa. Tavoitteena oli saada verkossa toimiva järjestelmä, joka olisi heti käyttöönotettavissa. Tämä myös toteutui.

Sovelluksen kehitys jaettiin osa-alueisiin. Viikko viikolta järjestelmään lisättiin toiminnallisuutta, sen mukaisesti kuin toimeksiantaja näitä toivoi. Kun ensimmäinen toimiva versio sovelluksesta oli valmis, siirryttiin lokaalista ympäristöstä myös virtuaalikoneelle, eli tämä vastaa käytännössä varsinaista käyttöönottoa. Tällöin ilmeni, että Angular 2-projektin aiempi konfigurointi ei tähän soveltunut, vaan projekti piti konfiguroida uusiksi, johon aikaa meni melko paljon. Tätä en osannut projektia aloittaessani ennakoida.

Kaikki sovelluksen vaatimukset toteutuivat. Sovellus myös toimi myös lähes täysin toivotulla tavalla. Ongelmia aiheutti ennakoiva tekstinsyöttö, johon aikaa meni erittäin paljon, eniten aikaa kaikista osakokonaisuuksista. Kokeilin viittä eri kolmannen osapuolen moduulia, joista ei mikään vastannut omia tarpeita. Päädyin kuitenkin valitsemaan yhden moduulin, joka vastasi eniten omia tarpeita. Joitakin ominaisuuksia sain muokattua sellaiseksi kuin toivoin, esimerkiksi, että käyttäjä voi tehdä valinnan painamalla näppäimistössä olevaa enter-näppäintä, hiiren valintanapin lisäksi, ja kursori jää valintalaatikkoon valinnan jälkeen useamman oireen haussa. Tarkoituksena kuitenkin oli, että ennakoiva tekstinsyöttö hakisi aina jokaisen sanan (vaikka oire tai sairaus olisi monisanainen) perusteella. Joko sain moduulin muokattua sellaiseksi, että se haki vain ensimmäisen sanan perusteella, tai haku suoritetaan siten, että kirjainyhdistelmät ilmenevät jossain kohtaa sanoissa. Päädyin käyttämään jälimmäistä vaihtoehtoa, välttääkseni sen, että joitakin oireita tai sairauksia jäisi kokonaan pois hakutuloksista. Moduulissa on käytetty RegExiä, mutta varmasti moduulin rakenteesta johtuen, muokkaukset RegExissä eivät vaikuttanut mitenkään moduulin toimintaan. Ongelma siis johtui jostain muusta. Tämä ongelma ei ratkennut lukuisista yrityksistä huolimatta, joten asia jäi myös hieman vaivaamaan.

Kun sovellus oli melkein valmis, huomattiin, että koska sovellus oli silloin täysin yksisivuinen, jossa vain piiloitettiin tai näytettiin sovelluksen elementtejä, eli html-koodin div-elementtejä, rippuen siitä mitä valintoja käyttäjä tekee sovelluksessa. Tämä kuitenkin

tarkoitti sitä, että selaimen takaisin- (tai eteenpäin)-painike ei reitittänyt käyttäjää edelliseen näkymään. Tässä vaiheessa lisäsin sovellukseen lapsireitityksiä ja -komponentteja, jotta näitä selaimessa olevia painikkeita olisi mahdollista käyttää sovelluksessa. Tämä kuitenkin tarkoitti sitä, että reitityksien ja komponenttien luomisen lisäksi lähdekoodia joutui myös muuttamaan jonkun verran. Syy tähän on se, että sovelluksessa verkkosivu ladataan asynkronisesti, eli reitityksen tapahtuessa html-näkymä tulostuu selaimeen ennen kuin tietokannasta on ehditty noutaa dataa, jolloin ilmenee virheitä ja sovellus ei näin ollen toimi. Jossain osissa pystyin käyttämään elvis-operaattoria, jossain osissa piti luoda täysin uusia muuttujia saadakseni sovelluksen toimimaan toivotulla tavalla.

7 POHDINTA

Sovelluksen jakaminen osakokonaisuuksiin oli jälkeenpäin katsottuna hyvä ratkaisu. Viikko viikolta sovellukseen lisättiin toiminnallisuutta. Sovelluksen suunnitteluun olisi voinut mahdollisesti panostaa hieman enemmän, koska aina kuin lisäsin sovellukseen toiminnallisuutta, sain olemassa olevaa lähdekoodia muokata, joskus vähemmän, joskus enemmän. Tämä oli ajoittain hieman turhauttavaa, kun tuntui, että tähän niin sanottuun ”virheiden” korjaamiseen meni ajoittain todella paljon aikaa. Sovelluksen vaatimukset olivat kuitenkin alusta asti tiedossa, olisi siis ollut mahdollista ennakoida miten lähdekoodia olisi kannattanut muokata. Toisaalta kokemusta ei aloittaessani ollut juuri ollenkaan- eli olisi ollut erittäin vaikeaa arvioida millaiseksi lähdekoodi kannattaisi toteuttaa, jotta myöhemmin lisätty toiminnallisuus tai ominaisuus integroidaan helposti olemassa olevaan lähdekoodiin, johtuen siitä, että sovelluksen kehittäminen oli alusta asti uusien asioiden oppimista. Vaikka suunnitteluun olisi käyttänytkin runsaasti aikaa, arvioin, että olemassa olevaa lähdekoodia olisi kuitenkin joutunut muokkaamaan. Kun työkokemusta kertyy lisää, osaa varmasti paremmin arvioida millaiseksi lähdekoodi kannattaa ennakoivasti toteuttaa, jotta uusia ominaisuuksia voi helposti integroida.

Sovelluksen toteuttaminen oli yllättäen helpompaa kuin olin alkuun ajatellut. Eniten ongelmia aiheutti edellisessä luvussa mainitsemani ennakoiva tekstinsyöttö, toiminnallisuuksien toteuttamisen sijaan.

Sovellus on valmis käyttöönotettavaksi, sovelluksen ulkoasu pitää kuitenkin muokata sellaiseksi kuin toimeksiantajaa sen haluaa. Nykyiseen ulkoasuun ei ole paljon panostettu, juuri yllä mainitun asian takia. Ulkoasu on vain tehty sellaiseksi, että ulkoasu on siisti ja sovelluksen osat jäsennelty.

Tarkoituksena on vielä jatkokehittää sovellusta, lähinnä tietokannan puolelta. Tauluja lisätään tietokantaan, esimerkiksi erillinen taulu sairauksien synonyymeille. Esimerkiksi voitaisiin ottaa, että ”diseases”-taulussa on sairaus ”kuume” ja synonyymitaulussa on vaikka kuumeelle synonyyminä ”lämpö”, ja nämä siis viittaisivat sovelluksessa kuitenkin yhteen ja samaan sairauteen. Tämän lisäksi on tarkoituksena luoda myös tauluja eri kielille, jotta sovelluksen kääntäminen eri kielille olisi mahdollisimman helppoa ja nopeaa.

Sovellusta voidaan myös kehittää eteenpäin esimerkiksi lisäämällä vielä kenttiä, jos vaikka halutaan rakentaa sovellukseen vielä enemmän logiikkaa sairauksien ja oireiden ja näiden välisten liitoksien ympärille. Näitä ajatellen olen myös yrittänyt muokata lähdekoodia sillä tavalla, että kenttien lisäys olisi mahdollisimman helppoa.

Kaiken kaikkiaan tämän sovelluksen kehittäminen on ollut opettavainen kokemus, jossa on oppinut paljon uusia asioita, joista on paljon hyötyä tulevaisuudessa.

LÄHTEET

4Dev 2016. Viitattu 11.10.2016. Integrating and Angular2 app with PHP app. http://4dev.tech/2016/07/using-http-client-to-integrate-angular2-to-php/

Angular 2016. Viitattu 5.10.2016. Quickstart. https://angular.io/docs/ts/latest/quickstart.html

Angular Blogspot 2016. Viitattu 5.10.2016. Blog. http://angularjs.blogspot.fi/2016/09/angular2-final.html

Lerner, A., Coury, F., Murray, N., Taborda, C. 2016. ng-Book 2. Revision 40.

Jyväskylän yliopisto. 2016. Relaatiotietokantojen peruskäsitteet. IT-tiedekunta ja avoin yliopisto. Viitattu 10.10.2016. http://appro.mit.jyu.fi/doc/tiedonhallinta/tietokannat/index2.html

Mainiotech 2016. Viitattu 4.12.2016. PHP. https://www.mainiotech.fi/tekniikat/php

Node.js 2016. Viitattu 30.11.2016. About Node.js. https://nodejs.org/en/about/

Ohjelmointiputka 2016. Viitattu 10.10.2016. MySQL ja PHP. http://www.ohjelmointiputka.net/oppaat/opas.php?tunnus=mysqlphp01

Ohjelmointiputka 2016. Viitattu 9.10.2016. PHP-ohjelmointi. http://www.ohjelmointiputka.net/oppaat/opas.php?tunnus=php_01

Php 2016. Viitattu 4.12.2016. Documentation. http://php.net/manual/en/intro-whatcando.php

PhpMyAdmin 2016. Viitattu 1.12.2016. Home. www.phpmyadmin.net

Pilvikoodarin Blogi 2016. Viitattu 30.11.2016. Node.js tutustumista. http://pilvikoodari.net/?p=67

Spring 2016. Viitattu 11.10. Understanding CORS. https://spring.io/understanding/cors

Stackoverflow 2016. Viitattu 4.12.2016. Angular 2 and MySQL concepts. http://stackoverflow.com/questions/34368897/angular-2-and-mysql-concepts

Stackoverflow 2009. Viitattu 30.11.2016. Asynchonous vs synchronous execution, what does it really mean?. http://stackoverflow.com/questions/748175/asynchronous-vs-synchronous-execution-what-does-it-really-mean

Stackoverflow 2016. Viitattu 11.10.2016. Can I start using Angular 2 without node?. http://stackoverflow.com/questions/39108403/can-i-start-using-angular-2-without-node

Suehring, S., Valade, J. 2013. PHP, MySQL, JavaScript & HTML5 All-in-One For Dummies. Hoboken, New Jersey.

TechTerms 2013. Viitattu 1.12.2016. WAMP. http://techterms.com/definition/wamp

Tutorialspoint 2016. Viitattu 9.12.2016. Angular 2 Tutorial. http://dev.tutorialspoint.com/angular2

Tutorialspoint 2016. Viitattu 30.11.2016. Node.js Tutorial, Introduction. https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm

Uniform Server 2016. Viitattu 1.12.2016. The Uniform Server Zero XI. http://www.uniformserver.com/ZeroXI_documentation/

W3schools 2016. Viitattu 9.10.2016. PHP. http://www.w3schools.com/php/php_syntax.asp

W3schools 2016. Viitattu 9.10.2016. PHP MySQL Database. http://www.w3schools.com/php/php_mysql_intro.asp

W3schools 2016. Viitattu 4.12.2016. PHP Introduction. http://www.w3schools.com/php/php_intro.asp

W3Tech 2016. Viitattu 5.12.2016. Usage of server-side programming languages for websites. https://w3techs.com/technologies/overview/programming_language/all¨